Numerical recovery strategies for parallel resilient Krylov linear solvers

E. Agullo* A. Guermouche† L. Giraud* J. Roman* M. Zounon*

May 26, 2016

Abstract

As the computational power of high performance computing (HPC) systems continues to increase by using a huge number of cores or specialized processing units, HPC applications are increasingly prone to faults. In this paper, we present a new class of numerical fault tolerance algorithms to cope with node crashes in parallel distributed environments. This new resilient scheme is designed at application level and does not require extra resources, i.e., computational unit or computing time, when no fault occurs. In the framework of iterative methods for the solution of sparse linear systems, we present numerical algorithms to extract relevant information from available data after a fault, assuming a separate mechanism ensures the fault detection. After data extraction, a well chosen part of missing data is regenerated through interpolation strategies to constitute meaningful inputs to restart the iterative scheme. We have developed these methods, referred to as Interpolation-Restart techniques, for Krylov subspace linear solvers. After a fault, lost entries of the current iterate computed by the solver are interpolated to define a new initial guess to restart the Krylov method. A well suited initial guess is computed by using the entries of the faulty iterate available on surviving nodes. We present two interpolation policies that preserve key numerical properties of well-known linear solvers, namely the monotonic decrease of the A-norm of the error of the conjugate gradient or the residual norm decrease of GMRES. The qualitative numerical behavior of the resulting scheme have been validated with sequential simulations, when the number of faults and the amount of data losses are varied. Finally, the computational costs associated with the recovery mechanism have been evaluated through parallel experiments.

1 Introduction

One of the current challenge in high performance computing (HPC) is to increase the level of concurrency by using the largest number of resources operated at lower energy consumption. The use of these parallel resources at large scale leads to a significant decrease of the mean time between faults (MTBF) of HPC systems. To cope with these unstable situations, parallel applications have to be resilient, *i.e.*, be able to compute a correct output despite the presence of faults.

To guarantee fault tolerance, two classes of strategies are required. One for the fault detection and the other for fault correction. Faults such as computational node crashes are obvious to detect while silent faults may be challenging to detect. To cope with silent faults, a duplication strategy is commonly used for fault detection [42, 20] by comparing the outputs, while triple modular redundancy (TMR) is used for fault detection and correction [40, 36]. However, the additional computational resources required by such replication strategies may represent a severe penalty. Instead of replicating computational resources, studies [6, 39] propose a time redundancy model for fault detection. It consists in repeating computation twice on the same resource. The

*Inria, France
†Université Bordeaux, France

advantage of time redundancy models is the flexibility at application level; software developers can indeed select only a set of critical instructions to protect. Recomputing only some instructions instead of the whole application lowers the time redundancy overhead [26]. In some numerical simulations, data naturally satisfy well defined mathematical properties. These properties can be efficiently exploited for fault detection through a periodical check of the numerical properties during computation [10].

Checkpoint/restart is the most studied fault recovery strategy in the context of HPC systems. The common checkpoint/restart scheme consists in periodically saving data onto a reliable storage device such as a remote disk. When a fault occurs, a roll back is performed to the point of the most recent and consistent checkpoint. According to the implemented checkpoint strategy, all processes may perform the periodical record simultaneously. It is called a coordinated checkpoint [12, 35, 29]. In parallel distributed environments, synchronizations due to coordination may significantly degrade application performance [13, 24]. To avoid synchronization, uncoordinated checkpoint may be employed combined with message logging protocols [3, 7, 21]. Many mechanisms have been developed to lower the overhead of the checkpoint/restart strategy [23, 30, 41]. However, the additional usage of resources (such as memory, disk) that is required by checkpoint/restart schemes may be prohibitive, or the time to restore data might become larger than the MTBF [9].

Algorithm based fault tolerance (ABFT) is a class of approaches in which algorithms are adapted to encode extra data for fault tolerance at expected low cost [18, 5, 16, 28]. The basic idea consists in maintaining consistency between extra encoded data and application data. The extra encoded data can be exploited for fault detection and for loss data recovery. ABFT strategies may be excellent candidates to ensure the resilience of an application; however they induce extra costs for computing and storing the data encoding even when no fault occurs.

In this paper, we present numerical resilient methods for linear system solvers that are the innermost numerical kernels in many scientific and engineering applications and also often ones of the most time consuming parts. To solve systems of linear equations, direct methods based on matrix decompositions, are commonly used because they are very robust. However, to solve large and sparse systems of linear equations, direct methods may require a prohibitive amount of computational resources (memory and CPU). To overcome the drawbacks of direct methods, iterative methods constitute an alternative widely used in many engineering applications. The basic idea is to approximate the solution of large sparse systems of linear equations, through successive iterations that require less storage and fewer floating-point operations. In addition to having attractive computational features for solving large sparse systems of linear equations, iterative methods are potentially more resilient. After a "perturbation" induced by a fault, the computed iterate can still serve as an initial guess as long as the key data that define the problem to solve, that are the matrix and the right-hand side, are not corrupted. We exploit this natural resilience potential to design robust resilience iterative solvers which may still converge in the presence of successive and possibly numerous faults.

In the context of parallel distributed computing, common faults are hardware node crashes. To cope with such node crashes often referred to as hard faults, an interpolation strategy has been introduced for GMRES in [22]. It consists in computing meaningful values for the lost entries of the current iterate through the solution of a relatively small linear system. The recovered iterate is then used as a new initial guess to restart GMRES. We name Linear Interpolation this class of methods and denote it LI in the sequel. Although this approach is suitable for iterative solvers in general, it can be applied only if the matrix associated with the local linear system to be solved for performing the interpolation is non singular. This property is guaranteed if the (global) matrix is SPD, but may not hold otherwise. For this reason, we introduce a new strategy that can be applied to any type of linear system. This alternative interpolation approach is based on a linear least squares solution that ensures the existence and uniqueness of the regenerated entries without any assumption on the matrix associated with the linear system. We name Least Square Interpolation this class of more robust but potentially more expensive methods and denote it LSI in the sequel. Furthermore, we generalize all considered LI and LSI techniques to simultaneous

multiple fault cases (i.e. when multiple nodes crash during the same iteration). In addition, using simple linear algebra arguments, we show that the developed techniques preserve key monotonic properties of CG and GMRES. In the sequel, these LI and LSI numerical resilient strategies are called Interpolation-Restart (IR) strategies.

The remaining of the paper is organized as follows. In Section 2, we present various IR techniques and analyze their numerical properties. Multiple fault cases are also discussed and we describe different approaches to handle them. We consider the two main Krylov subspace methods, namely CG and GMRES and demonstrate that the proposed IR strategies preserve their key numerical properties. We particularly focus on variants of preconditioned GMRES and discuss how the location of the preconditioner impacts the properties of our IR strategies. Section 3 is devoted to the qualitative numerical study of the resilient schemes. The fault rate and the volume of damaged data are varied to study the robustness of the IR strategies. In Section 4, we present parallel designs for the considered IR strategies that we use for illustrating their respective computational cost in a parallel distributed environment. Finally, some conclusions and perspectives are discussed in Section 5.

2 Strategies for resilient iterative methods

2.1 Context

In this section, we introduce the governing ideas that underlie the design of the IR strategies. For the sake of exposure, we restrict ourselves to parallel distributed environments although these strategies can be extended to other HPC contexts as long as a reliable fault detection mechanism is available. On parallel distributed platforms, the crash of a node is usually easily detected. We consider the solution of sparse linear system

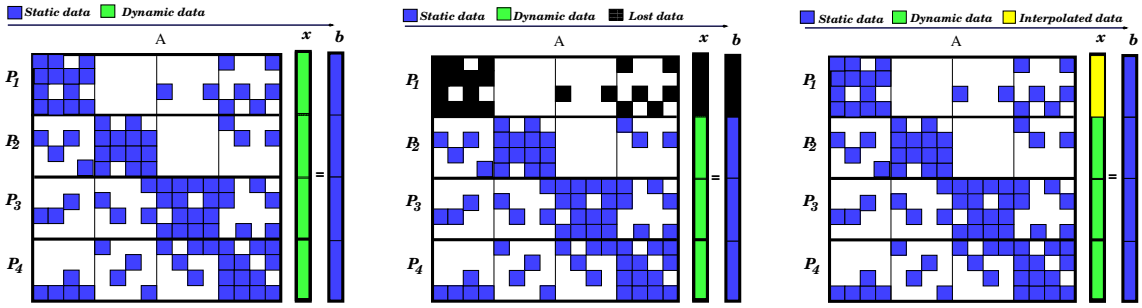
$$Ax = b,$$

where the matrix $A \in \mathbb{R}^{n\times n}$ is nonsingular, the right-hand side $b \in \mathbb{R}^n$ and the solution $x \in \mathbb{R}^n$.

Assumption 1. *In our parallel computational context, all the vectors or matrices of dimension n are distributed by blocks of rows in the memory of the different computing nodes but scalars or low dimensional matrices are replicated.*

According to Assumption 1, the right-hand side b and the coefficient matrix A are distributed according to a block-row partition as well as all vectors of dimension n generated during the solve step whereas scalars or low dimensional matrices are replicated on all nodes. Let N be the number of partitions, such that each block-row is mapped to a computing node. For all p, $p \in [1, N]$, I_p denotes the set of row indices mapped to node p. With respect to this notation, node p stores the block-row $A_{I_p,:}$ and x_{I_p} as well as the entries of all the vectors involved in the solver associated with the corresponding row indices of this block-row. If the block A_{I_p,I_q} contains at least one nonzero entry, node p is referred to as neighbor of node q as communication will occur between those two nodes to perform a parallel matrix-vector product. By $J_p = \{\ell, a_{\ell,I_p} \neq 0\}$, we denote the set of row indices in the block-column $A_{:,I_p}$ that contain nonzero entries and $|J_p|$ denotes the cardinality of this set.

When a node crashes, all available data in its memory are lost. We consider the formalism proposed in [22] in the same computing framework, where data loss are classified into three categories: *computational environment*, *static* data and *dynamic* data. The computational environment is all data needed to perform the computation (code of the program, environment variables, ...). Static data are those that are setup during the initialization phase and that remain unchanged during the computation. They correspond to the input data to the problem and include in particular the coefficient matrix A, the right-hand side vector b. Dynamic data are all data whose value may change during the computation. The Krylov basis vectors (e.g., Arnoldi basis, descent directions,



(a) Before Fault (b) Faulty iteration (c) Interpolation

Figure 1: General interpolation scheme. The matrix is initially distributed with a block-row partition, here on four nodes (a). When a fault occurs on the node P_1, the corresponding data are lost (b). Whereas static data can be immediately restored, dynamic data that have been lost cannot and we investigate numerical strategies for regenerating them (c).

residual, ...) and the iterate are examples of dynamic data. In Figure 1a, we depict a block row distribution on four nodes. In blue, we have static data associated with the linear system (i.e., matrix and right-hand side) while dynamic data (here only the iterate is shown) are in green. If the node P_1 fails, the first block-row of A as well as the first entries of x and b are lost (in black in Figure 1b). Because our primary interest is in the numerical behavior of the schemes, we possibly make strong assumptions on the parallel environment. In particular, we assume that when a fault occurs, the crashed node is replaced and the associated computational environment and static data are restored [22]. In Figure 1c for instance, the first matrix block-row as well as the corresponding right-hand side are restored as they are static data. However, the iterate being dynamic is definitely lost and must be recovered. For the sake of generality among Krylov methods, we do not attempt to regenerate all the dynamic data but only the iterate. Our approach consists in interpolating the lost entries of the iterate using interpolation strategies adapted to the linear systems to be solved. The interpolated entries and the current values available on the other nodes define the recovered iterate which is hence used as an initial guess to restart the Krylov method.

We assume in the rest of Section 2 that the fault occurs during iteration k+1 and the interpolation strategies are based on the values of the iterate at iteration k. Firstly, we assume that only one fault occurs at a time in Sections 2.2 and 2.3. After, we relax that assumption in Section 2.4 to consider simultaneous multiple fault cases (at the iteration granularity).

2.2 Linear interpolation

The LI strategy, first introduced in [22], consists in interpolating lost data by using data from surviving nodes. Let $x^{(k)}$ be the approximate solution when a fault occurs. After the fault, the entries of $x^{(k)}$ are known on all nodes except the failed one. The LI strategy computes a new approximate solution by solving a local linear system associated with the failed node. If the node p fails, $x^{(LI)}$ is computed via

$$\begin{cases} x^{(LI)}_{I_q} = x^{(k)}_{I_q} & \text{for } q \neq p, \\ x^{(LI)}_{I_p} = A^{-1}_{I_p,I_p}(b_{I_p} - \sum_{q \neq p} A_{I_p,I_q} x^{(k)}_{I_q}). \end{cases} \tag{1}$$

The motivation for this interpolation strategy is that, at convergence (i.e., $x^{(k)} = x$), it regenerates the conservative solution ($x^{(LI)} = x$) as long as A_{I_p,I_p} is non-singular. Furthermore we show that such an interpolation exhibits a property in term of A-norm of the error for symmetric positive definite (SPD) matrices as expressed in the proposition below.

Proposition 1. *Let A be SPD. Let $k+1$ be the iteration during which node p crashes. The regenerated entries $x_{I_p}^{(LI)}$ defined by Equation (1) are always uniquely defined. Furthermore, let $e^{(k)} = x - x^{(k)}$ denote the forward error associated with the iterate before the fault occurs, and $e^{(LI)} = x - x^{(LI)}$ be the forward error associated with the recovered iterate computed using the LI strategy* (1)*, we have:*

$$\|e^{(LI)}\|_A \leq \|e^{(k)}\|_A.$$

Proof. 1. Uniquely defined $x_{I_p}^{(LI)}$: because A is SPD so is A_{I_p,I_p} that is consequently non-singular.

2. Monotonic decrease of $\|e^{(LI)}\|_A$: for the sake of simplicity of exposure, but without any loss of generality we consider a two node case and assume that the first node crashes. Let $A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ be a SPD matrix, where $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ denotes the exact solution of the linear solution. The equations associated with the exact solution are:

$$\begin{aligned} A_{1,1}x_1 + A_{1,2}x_2 &= b_1, \\ A_{2,1}x_1 + A_{2,2}x_2 &= b_2. \end{aligned} \tag{2a}$$

By linear interpolation (Equation (1)), we furthermore have:

$$A_{1,1}x_1^{(LI)} + A_{1,2}x_2^{(k)} = b_1, \tag{3a}$$

$$x_2^{(LI)} = x_2^{(k)}. \tag{3b}$$

The proof consists in showing that $\delta = \|x^{(LI)} - x\|_A^2 - \|x^{(k)} - x\|_A^2$ is non-positive. Given two vectors, y and z, we recall that:

$$y^T A z = y_1^T A_{1,1} z_1 + y_1^T A_{1,2} z_2 + y_2^T A_{2,1} z_1 + y_2^T A_{2,2} z_2,$$

$$\begin{aligned} \|y\|_A^2 &= y_1^T A_{1,1} y_1 + y_2^T A_{2,2} y_2 + 2 y_1^T A_{1,2} y_2, \\ \|y - z\|_A^2 &= y^T A y - 2 y^T A z + z^T A z, \qquad (4) \\ (y+z)^T A (y - z) &= y^T A y - z^T A z. \qquad (5) \end{aligned}$$

The proof consists in showing that $\delta = \|x^{(LI)} - x\|_A^2 - \|x^{(k)} - x\|_A^2$ is non-positive.
It is easy to see by (3b) and (4) that:

$$\begin{aligned} \delta = (x_1^{(LI)})^T \left(A_{1,1} x_1^{(LI)} + 2 A_{1,2} x_2^{(k)} \right) - (x_1^{(k)})^T \left(A_{1,1} x_1^{(k)} + 2 A_{1,2} x_2^{(k)} \right) \\ + 2 \left((x_1^{(k)})^T A_{1,1} x_1 + (x_1^{(k)})^T A_{1,2} x_2 - (x_1^{(LI)})^T A_{1,1} x_1 - (x_1^{(LI)})^T A_{1,2} x_2 \right). \end{aligned}$$

By (2a) and (5), we have:

$$\begin{aligned} \delta &= \left(x_1^{(LI)} - x_1^{(k)} \right)^T A_{1,1} \left(x_1^{(LI)} + x_1^{(k)} \right) + 2 \left(x_1^{(LI)} - x_1^{(k)} \right)^T \left(A_{1,2} x_2^{(k)} - b_1 \right) \\ &= \left(x_1^{(LI)} - x_1^{(k)} \right)^T \left(A_{1,1} x_1^{(LI)} + A_{1,2} x_2^{(k)} - 2 b_1 + A_{1,1} x_1^{(k)} + A_{1,2} x_2^{(k)} \right). \end{aligned}$$

Because A is SPD, so is $A_{1,1}$ and $A_{1,1}^T A_{1,1}^{-1} = I$. Then by (3a), we have,

$$\begin{aligned}
\delta &= \left(x_1^{(LI)} - x_1^{(k)}\right)^T A_{1,1}^T A_{1,1}^{-1} \left(-b_1 + A_{1,1}x_1^{(k)} + A_{1,2}x_2^{(k)}\right) \\
&= -\left((A_{1,1}x_1^{(LI)}) - (A_{1,1}x_1^{(k)})\right)^T A_{1,1}^{-1} \left(b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)}\right), \\
&= \left(b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)}\right)^T A_{1,1}^{-1} \left(b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)}\right), \\
&= -\|b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)}\|^2_{A_{1,1}^{-1}} \\
&\leq 0.
\end{aligned}$$

Note that this proof gives us also a quantitative information on the error decrease.

□

In the general case (i.e., non-SPD), it can be noticed that the LI strategy is only defined if the diagonal block A_{I_p,I_p} has full rank. In the next section, we present a new interpolation variant that does not make any rank assumption and enable more flexibility in multiple fault cases.

2.3 Least squares interpolation

The LI strategy is based on the solution of a local linear system. The new variant we propose, LSI, relies on a least squares solution. Assuming that the node p has crashed, x_{I_p} is interpolated as follows:

$$\begin{cases} x_{I_q}^{(LSI)} = x_{I_q}^{(k)} & \text{for } q \neq p, \\ x_{I_p}^{(LSI)} = \underset{x_{I_p}}{\operatorname{argmin}} \|(b - \sum_{q \neq p} A_{:,I_q} x_q^{(k)}) - A_{:,I_p} x_{I_p}\|. \end{cases} \tag{6}$$

We notice that the matrix involved in the least squares problem, $A_{:,I_p}$, is sparse of dimension $|J_p| \times |I_p|$ where its number of rows $|J_p|$ depends on the sparsity structure of $A_{:,I_p}$. Consequently the LSI strategy has a higher computational cost, but it overcomes the rank deficiency drawback of LI because the least squares matrix is always full column rank (as A is full rank).

Proposition 2. *Let $k+1$ be the iteration during which the fault occurs on the node p. The regenerated entries of $x_{I_p}^{(LSI)}$ defined in Equation (6) are uniquely defined. Furthermore, let $r^{(k)} = b - Ax^{(k)}$ denote the residual associated with the iterate before the fault occurs, and $r^{(LSI)} = b - Ax^{(LSI)}$ be the residual associated with the recovered iterate generated with the LSI strategy* (6)*, we have:*

$$\|r^{(LSI)}\|_2 \leq \|r^{(k)}\|_2.$$

Proof. 1. Uniquely defined: because A is non-singular, $A_{:,I_p}$ has full column rank.

2. Monotonic residual norm decrease: the proof is a straightforward consequence of the definition of $x_{I_p}^{(LSI)} = \underset{x_{I_p}}{\operatorname{argmin}} \|(b - \sum_{q \neq p} A_{:,I_q} x_q^{(k)}) - A_{:,I_p} x_{I_p}\|$.

□

Remark 1. *Notice that the LSI technique preserves the true solution. This means that if the fault occurs at the iteration where the stopping criterion based on a scaled residual norm is satisfied, the recovered iterate regenerated by LSI also complies with the stopping criterion.*

2.4 Multiple faults

In the previous section, we have introduced two policies to handle a single fault occurrence. Although the probability of this event is very low, multiple faults may occur during the same iteration especially when a huge number of nodes is used. At the granularity of our approach, these faults may be considered as simultaneous. If for example, two nodes p and q crash simultaneously but they are not neighbor, then x_{I_p} and x_{I_q} can be regenerated independently. In this section, we focus more precisely on simultaneous faults on neighbor nodes. To deal with such multiple faults, we present two new approaches based on IR strategies.

2.4.1 Global interpolation techniques

We consider here a strategy consisting in handling multiple faults all at once. With this global interpolation technique, the linear system is permuted so that the equations relative to the crashed nodes are grouped into one block. Therefore the interpolation technique falls back to the single fault case. For example, if nodes p and q crash, this global linear interpolation (LI-G) solves the following linear system (similar to Equation (1))

$$\begin{pmatrix} A_{I_p,I_p} & A_{I_q,I_p} \\ A_{I_q,I_p} & A_{I_q,I_q} \end{pmatrix} \begin{pmatrix} x_{I_p}^{(LI-G)} \\ x_{I_q}^{(LI-G)} \end{pmatrix} = \begin{pmatrix} b_{I_p} - \sum_{\ell \notin \{p,q\}} A_{I_p,I_\ell} x_{I_\ell}^{(k)} \\ b_{I_q} - \sum_{\ell \notin \{p,q\}} A_{I_q,I_\ell} x_{I_\ell}^{(k)} \end{pmatrix}.$$

Following the same idea, the global least squares interpolation (LSI-G) solves

$$\begin{pmatrix} x_{I_p}^{(LSI-G)} \\ x_{I_q}^{(LSI-G)} \end{pmatrix} = \underset{x_{I_p}, x_{I_q}}{\operatorname{argmin}} \| (b - \sum_{\ell \notin \{i,j\}} A_{:,I_\ell} x_\ell^{(k)}) - A_{(:,I_p \cup I_q)} \begin{pmatrix} x_{I_p} \\ x_{I_q} \end{pmatrix} \|.$$

2.4.2 Local interpolation techniques

Alternatively, if neighbor nodes p and q crash simultaneously, x_{I_p} and x_{I_q} can be interpolated independently from each other. Using the LI strategy, the entries of x_{I_p} can be computed using Equation (1) assuming that the quantity x_{I_q} is equal to its initial value $x_{I_q}^{(0)}$. At the same time, node q regenerates x_{I_q} assuming that $x_{I_p} = x_{I_p}^{(0)}$. We call this approach uncorrelated linear interpolation (LI-U). For example we regenerate x_{I_p} via

1: $x_{I_q}^{(k)} = x_{I_q}^{(0)}$,
2: $x_{I_\ell}^{(LI-U)} = x_{I_\ell}^{(k)}$ for $\ell \notin \{p, q\}$,
3: $x_{I_p}^{(LI-U)} = A_{I_p,I_p}^{-1}(b_{I_p} - \sum_{p \neq q} A_{I_p,I_q} x_{I_q}^{(k)})$.

Although better suited for a parallel implementation, this approach might suffer from a worse interpolation quality when the off-diagonal blocks A_{I_p,I_q} or A_{I_q,I_p} define a strong coupling. Similar idea can be applied to LSI to implement an uncorrelated LSI (LSI-U). However, the flexibility of LSI can be further exploited to reduce the potential bad effect of using $x_{I_q}^{(0)}$ when regenerating x_{I_p}. Basically, to regenerate x_{I_p}, each equation that involves x_{I_q} is discarded from the least squares system and we solve the following equation

$$x_{I_p}^{(LSI-U)} = \underset{x_{I_p}}{\operatorname{argmin}} \| (b_{J_p \setminus J_q} - \sum_{\ell \notin \{p,q\}} A_{J_p \setminus J_q, I_\ell} x_{I_\ell}^{(k)}) - A_{J_p \setminus J_q, I_i} x_{I_p} \|, \tag{7}$$

where the set of row-column indices $(J_p \setminus J_q, I_\ell)$ denotes the set of rows of block column I_ℓ of A that have nonzero entries in row J_p and zero entries in row J_q (if the set $(J_p \setminus J_q, I_\ell) = \emptyset$ then $A_{J_p \setminus J_q, I_\ell}$ is a zero matrix).

We denote this approach by de-correlated LSI (LSI-D). The heuristic beyond this approach is to avoid perturbing the regeneration of x_{I_p} with entries in the right-hand sides that depend on x_{I_q} that are unknown. A possible drawback is that discarding rows in the least squares problem might lead to an under-determined or to a rank deficient problem. In such a situation, the minimum norm solution might be meaningless with respect to the original linear system. Consequently the recovered iterate might be poor to restart the Krylov method and could strongly penalize the overall convergence.

2.5 Numerical properties of the Interpolation-Restart Krylov methods

The CG method is commonly used for the solution of linear systems involving SPD matrices [17]. The CG algorithm enjoys the unique property to minimize the A-norm of the forward error on the Krylov subspaces, i.e., $\|x^{(k)} - x\|_A$ is monotonically decreasing along the iterations k (see for instance [33]). This decreasing property is still valid for the preconditioned conjugate gradient (PCG) method. Consequently, an immediate consequence of Proposition 1 reads:

Corollary 1. *The recovered iterate generated by either LI or LI-G after a single or a multiple node crash does ensure that the A-norm of the forward error associated with the IR strategy is monotonically decreasing for CG and PCG.*

The GMRES method is one of the most popular solver for the solution of non-symmetric linear systems. It belongs to the class of Krylov solvers that minimize the 2-norm of the residual associated with the iterates built in the sequence of Krylov subspaces (MINRES is another example of such a solver [27]). In contrast to many other Krylov methods, GMRES does not update the iterate at each iteration but only either when it has converged or when it restarts every other m steps in the so-called restarted GMRES (GMRES(m)) [34]. When a node crashes, the approximate solution is not available. According to Assumption 1, the Hessenberg matrix is replicated on each node and the least squares problem is also solved redundantly. Consequently, each individual still running node ℓ can compute its entries I_ℓ of the iterate when a fault occurs. The property of residual norm monotonic decrease of full and restarted GMRES is still valid in case of fault for the IR strategies LSI (for single fault) and LSI-G (even for multiple faults).

Corollary 2. *The IR strategies LSI and LSI-G do ensure the monotonic decrease of the residual norm of minimal residual Krylov subspace methods such as GMRES, Flexible GMRES [32] and MINRES after a restart due to a fault.*

We should point out that this corollary does not translate to preconditioned GMRES as it is the case for PCG in Corollary 1. For preconditioned GMRES, the minimal residual norm property applies to the preconditioned residual. Consequently, to preserve this property after a fault, the linear problem to be solved should involve sub-matrices associated with the preconditioned systems AM or MA, where M is the preconditioner. For general preconditioner, these sub-matrices are not explicitly formed and cannot be used for the recovery. For left preconditioner, because GMRES solves the linear system $MAx = Mb$, we can still compute a recovered iterate $x^{(k)}$ using sub-matrices from A, but we loose the monotony property. For right preconditioned GMRES, $AMu = b$ with $x = Mu$, not much can be made except for block diagonal preconditioner (consequently local) where the property might still hold. Finally, the possible difficulties associated with general preconditioner for GMRES disappear when Flexible GMRES is considered. In that latter case, the generalized Arnoldi relation $AZ_k = V_{k+1}\bar{H}_k$ holds (using the classical notation from [32]), so that the still alive nodes can compute their part of x_k from their piece of Z_k.

3 Numerical experiments

In this section, we analyze the qualitative numerical behavior of the resilient Krylov solvers based on IR strategies. In Section 3.2, we present numerical experiments where at most one fault occurs

during one iteration. In contrast, section 3.3 presents examples where multiple faults occur during some iterations to illustrate the numerical robustness of the different variants exposed in Section 2.4. For the sake of completeness and to illustrate the possible numerical penalty induced by the restarting procedure after the faults, we compare in Section 3.4 the convergence behavior of the solvers with and without fault.

3.1 Experimental framework

For the sake of flexibility of the experiments, we have developed a simulator to monitor the amount of data lost at each fault as well as the rate of faults. Faults are injected using the Weibull probability distribution [43] that is supposed to be the most realistic probabilistic model that characterizes the normal behavior of large-scale computational platforms [31]. We vary some of its parameters to increase or decrease the number of faults for a given calculation. we control also the number of simulated nodes to vary the amount of data loss when a fault occurs. We have performed extensive numerical experiments and report only few examples that are representative of our observations (more experiments are available in the appendix of [1]). Most of the matrices come from the University of Florida (UF) test suite [11]. The right-hand sides are computed for a given solution generated randomly.

To study the numerical features of the proposed IR strategies, we display the convergence history as a function of the iterations, that also coincide with the number of preconditioned matrix-vector products. For the non-symmetric solver, we depict the scaled residual, while for the SPD case we depict the A-norm of the error. To distinguish between the interpolation quality effect and possible convergence delay introduced by the restart, we consider a simple strategy that consists in enforcing the restart after iteration k using $x^{(k)}$ as the initial guess (we do not inject faults, we only restart the solver at the iterations corresponding to faulty iterations observed during LI/LSI execution). We refer to this strategy as Enforced Restart and denote it ER. Furthermore, we also depict in red a straightforward strategy where the lost entries of the iterate are replaced by the corresponding ones of the first initial guess. This simple approach is denoted "*Reset*". For the sake of simplicity, the acronyms used to denote the names of different curves are recalled in the Table 1.

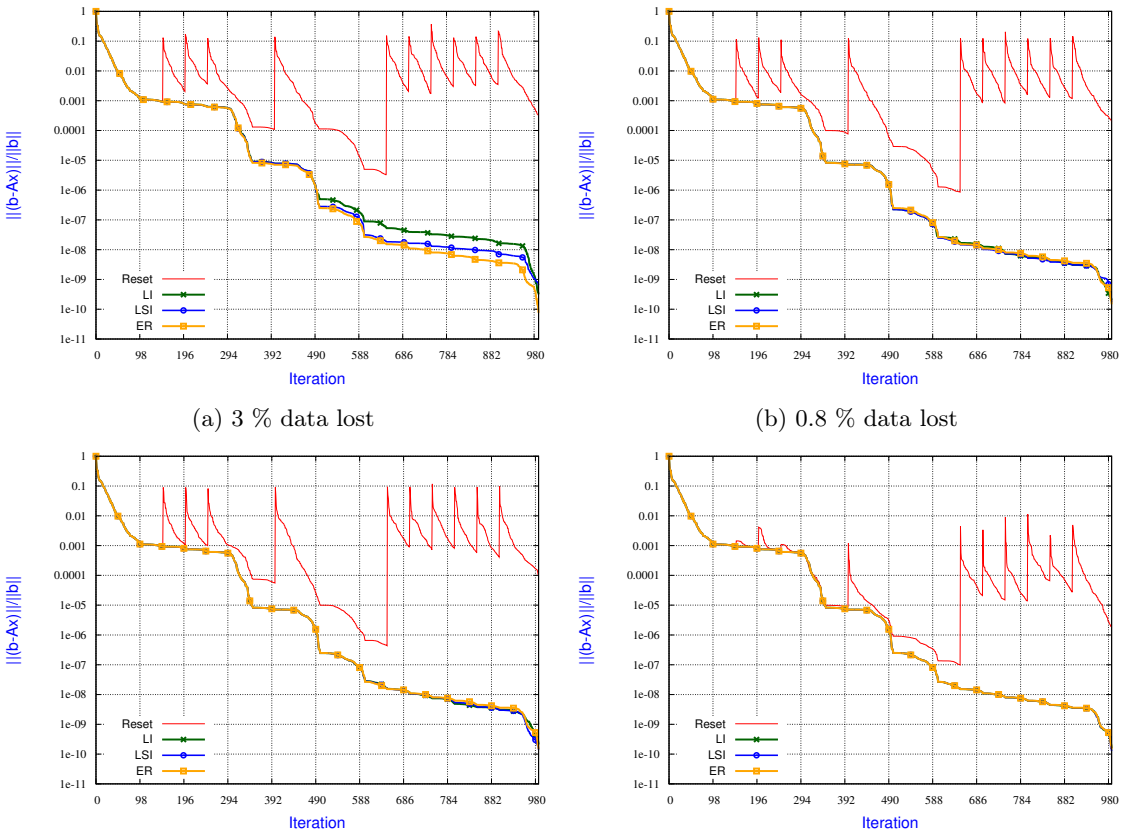
Acronym	Definition	Fault
Reset	Replace lost data by its initial value	Single/Multiple
LI	Linear interpolation	Single
LI-G	Global linear interpolation	Multiple
LI-U	Uncorrelated linear interpolation	Multiple
LSI	Least square interpolation	Single
LSI-G	Global least square interpolation	Multiple
LSI-U	Uncorrelated least square interpolation	Multiple
LSI-D	De-correlated least square interpolation	Multiple
ER	Enforced restart	Single/Multiple
NF	No faulty execution	–

Table 1: Definition of the acronyms used in the captions of forthcoming plots.

3.2 Numerical behavior in single fault cases

In this section, we examine the situation where only one fault occurs during an iteration. In Figure 2, the volume of lost entries varies from 3 % to 0.001 % (a single entry is lost in that latter case), at each fault. We report on experiments with GMRES(100) for the matrix Averous and refer to Table 1 for a short summary of the notations used in the legend. For these experiments,

in order to enable cross comparison, the number of faults is identical and they occur at the same iteration for all the runs. It can be observed that the straightforward restarting Reset policy does not lead to convergence. Each peak in the convergence history corresponds to a fault showing that the solver does not succeed to converge. In contrast, the IR strategies ensure convergence and have very similar convergence behavior. In a nutshell, they exhibit similar robustness capabilities. Furthermore, the convergence behavior of IR strategies is not much affected by the amount of data loss.



(a) 3 % data lost

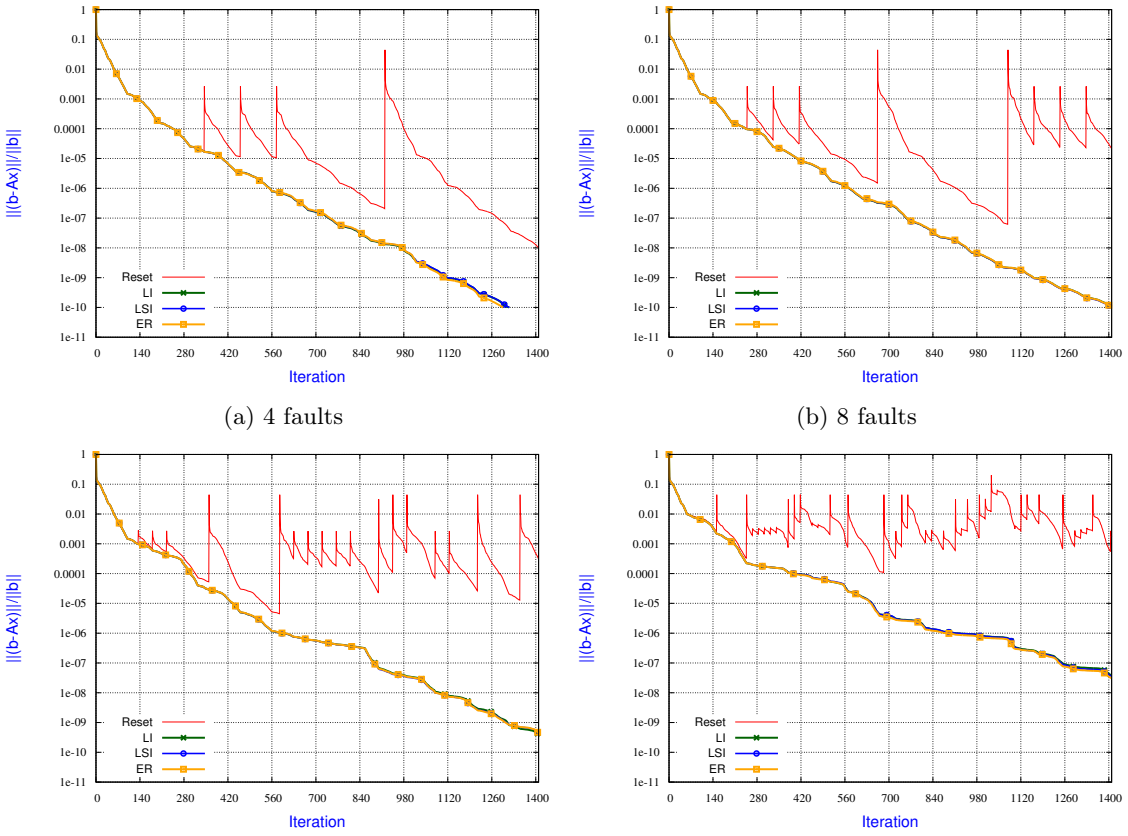
(b) 0.8 % data lost

(c) 0.2 % data lost

(d) 0.001 % data lost

Figure 2: Numerical behavior of the IR strategies when the amount of data loss is varied (matrix Averous/epb3 with 10 faults).

In Figure 3, we investigate the robustness of the IR strategies when the rate of faults is varied while the amount of recovered entries remains the same after each fault, that is 0.2 %. Those experiments are conduced with a GMRES(100) using the kim1 matrix. An expected general trend that can be observed on that example is: the larger the number of faults the slower the convergence. When only a few faults occur, the convergence penalty is not significant compared to the non-faulty case. For a large number of faults, the convergence is slowed down but continues to take place. For instance, for an expected accuracy of 10^{-7}, the number of iterations with 40 faults is twice the one without fault. Although not illustrated in the selected numerical experiments reported in Figure 2 and 3, the LI strategy failed in many of the experiments we ran because of the singularity of the A_{I_p,I_p} block. This constitutes a severe lack of robustness for this approach for non-SPD matrices. When LI does not fail, a general trend [1] is that none of the policies LI or LSI appears significantly and consistently superior to the other. The IR strategies based on either of both interpolation strategies have similar numerical behavior and are comparable with



(a) 4 faults

(b) 8 faults

(c) 17 faults

(d) 40 faults

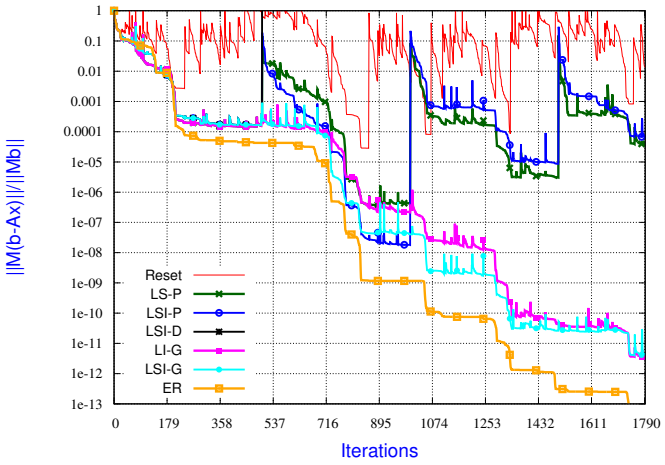
Figure 3: Numerical behavior of the IR strategies when the rate of faults varies (matrix Kim/kim1 with 0.2 % data loss at each fault)

ER. This comparison between the IR strategies and ER shows that the regenerated data are good approximation of lost data.

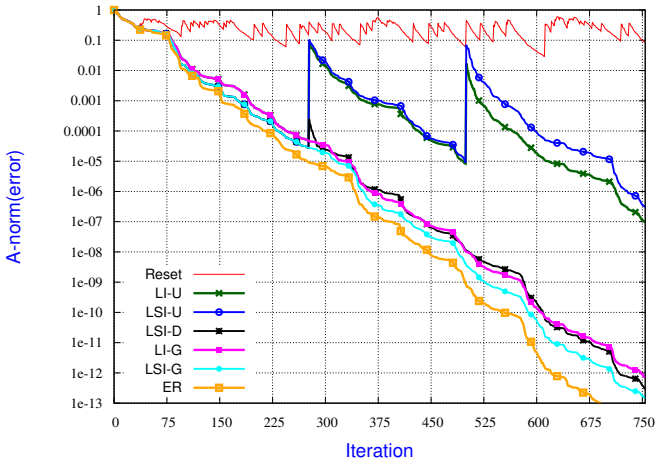
3.3 Numerical behavior in multiple fault cases

In this section, we illustrate the numerical behavior of the various IR strategies described in Section 2.4. We made a selection of a few numerical experiments and reported them in Figure 4. We recall that what is referred to as a multiple crash corresponds to the situation where two neighboring nodes crash during the same iteration, that is, the entries of x_{I_p} and x_{I_q} are lost at the same iteration and either the block A_{I_p,I_q} or the block A_{I_q,I_p} is nonzero (i.e., nodes p and q are neighbor). Furthermore, to be able to observe a few multiple faults using our fault injection probability law, we had to generate a very large number of faults. This situation has a very low probability to occur on real systems but deserves some observations on the numerical behavior of the interpolation schemes in such extreme situations.

In Figure 4, the multiple fault occurrences are characterized by a significant jump of the residual norm for GMRES and of the A-norm of the error for PCG for the two IR strategies LI-U and LSI-U, which are almost as poor as the straightforward Reset approach. The underlying idea to design these heuristics was to interpolate lost entries by fully ignoring other simultaneous faults (enabling a natural parallelism in the interpolation). Those experiments show that the penalty to pay is very high and that a special treatment deserves to be implemented.



(a) Left preconditioned GMRES (UF Averous/epb0 - 103 single and 3 multiple faults)



(b) PCG on a 7-point stencil (3D Reaction-diffusion equation - 67 single and 2 multiple faults)

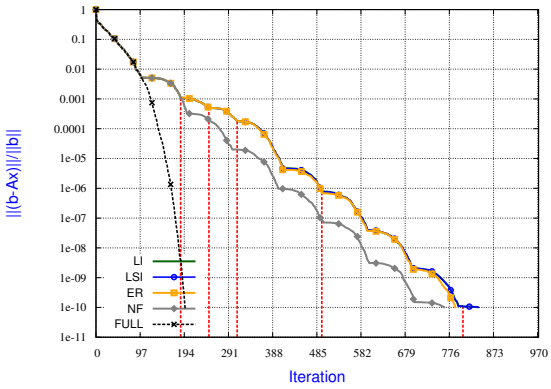
Figure 4: Numerical behavior of the IR strategies with multiple faults.

The first possibility is to consider the LI-G for SPD or the LSI-G for general matrices, where all the lost entries are regenerated at once as if a "large" single fault occurred. It can be seen in these figures that the numerical behavior is consequently very similar to the ones we observed in the previous section where only single faults were considered. More interesting is the behavior of the LSI-D strategy whose behavior seems to vary a lot from an example to another. In Figures 4b, this policy enables a convergence similar to the robust strategies LI-G and LSI-G, while in Figures 4a a jump is observed with this IR strategy (the convergence curve disappears from the plot area at iteration 530 and never shows up again because the preconditioned scaled residual remains larger than one). Actually, this latter bad behavior occurs when the least squares problem, that is solved once the correlated rows have been discarded, becomes rank deficient. In that case, the regenerated initial guess is extremely poor. In order to remove this drawback, one could switch to LI-G or LSI-G when a rank deficiency in the least squares matrix is detected. Such an hybrid scheme would conciliate robustness and speed of the IR approach and would thus certainly represent a relevant strategy for such extremely unstable computing environments.

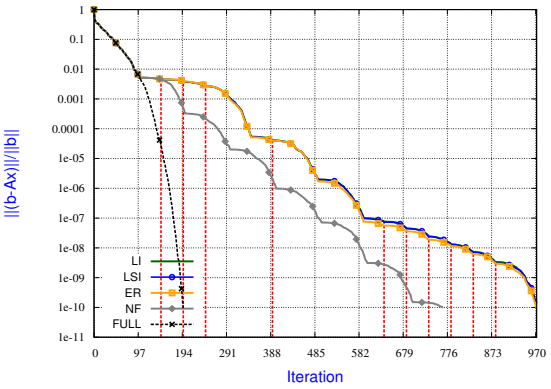
Remark 2. *In Figure 4a, some residual norm increases can be observed for the LSI-G variants with GMRES. This is due to the fact that after a fault, the least squares problems are built using sub-matrices of A (not of MA), consequently the assumption of Corollary 2 does not hold.*

3.4 Penalty of the Interpolation-Restart strategy on convergence

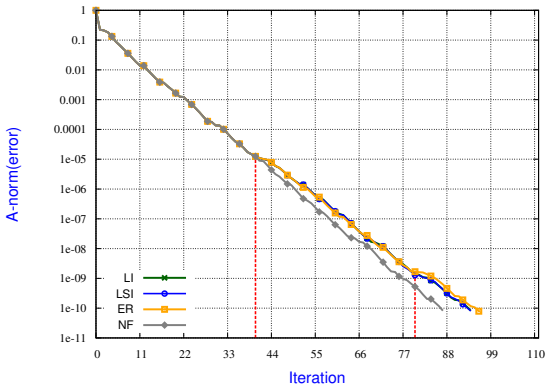
One of the main features of the resilient numerical algorithms described in this paper is to restart once meaningful entries have been interpolated to replace the lost ones. When restarting, the Krylov subspace built before the fault is lost and a new sequence of Krylov subspaces is computed. To reduce the computational resource consumption, such a restarting mechanism is implemented in GMRES that it is known to delay the convergence compared to full-GMRES. This delay can be observed in Figure 5a-5b, where the convergence history of full-GMRES is also depicted. Although the convergence history of the faulty executions are much slower than the one of full-GMRES, they are not that far and remain close to the non-faulty restarted GMRES(100).



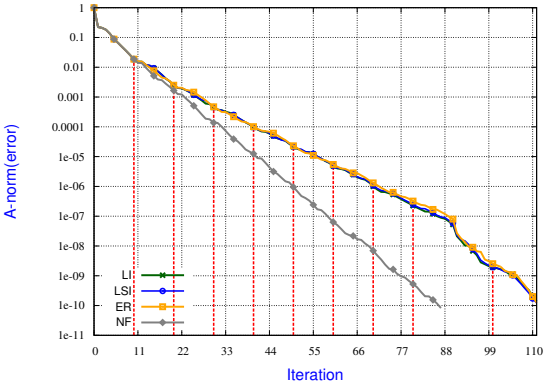
(a) GMRES(100) with matrix Chevron/Chevron2 with 5 faults



(b) GMRES(100) with matrix Chevron/Chevron2 with 10 faults



(c) PCG with matrix Cunningham/qa8fm 2 faults.



(d) PCG with with matrix Cunningham/qa8fm 9 faults

Figure 5: Convergence penalty induced by the restart after the interpolation (the faulty iterations are shown by a vertical dashed red line).

On the contrary, CG does not need to be restarted. In order to evaluate how the restarting affects the convergence of these two short-term recurrence solvers, we display in Figure 5c-5d the convergence history of CG of the method with and without fault. As already mentioned in Section 3.2 for experiments with GMRES, the larger the number of faults, the larger the convergence penalty.

4 Parallel experiments

In this section, we describe the extra data structures necessary for the implementation of IR strategies in a parallel distributed environment. In the same way, we present also the main computational and communication phases required for the parallel implementation. At the end, we report and analyze the parallel computational costs of the IR strategies using a few test matrices with a few hundreds of nodes.

For the sake of simplicity of exposure, let us assume that the matrices associated with the linear systems are symmetric in pattern. Consequently, they can be represented by a graph referred to as an adjacency graph. Let $G = \{V, E\}$ be the adjacency graph associated with A. In this graph, each vertex in V is associated with a row or column of the matrix A and it exists an edge in E between the vertices i and j if the entry $a_{i,j}$ is nonzero. In the sequel, to facilitate the exposure

and limit the notation we voluntarily mix a vertex of G with its index depending on the context of the description.

Our parallel implementations of the Krylov solvers rely on a block row partitioning of the matrix entries where the rows of indices I_p are mapped on node p. These sets of indices are defined by partitioning the graph G into N non-overlapping sub-graphs G_1^0, ..., G_N^0, where the set of indices of the vertices in G_p^0 is I_p. Let define G_i^1 as the one-overlap decomposition of G where $G_i^1 \supset G_i^0$ is obtained by including all the immediate neighboring vertices to those in G_i^0. By recursively applying this idea, the δ-overlap partition of G can be defined where the subsets are denoted G_i^δ.

4.1 Parallel implementation of LI

The LI strategy involves the off-diagonal entries of the row block $A(I_p, :)$, which are defined by $G_p^1 \backslash G_p^0$. The vertices of this subgraph define the entries of $x_{I_q}^{(k)}$ required, while the edges define the nonzero entries of A involved in the computation of right-hand side of Equation (1), that is $\sum_{q \neq p} A_{I_p, I_q} x_{I_q}^{(k)}$. To compute the right-hand side, the required communication and computation are similar to the ones involved in a regular sparse matrix-vector product performed at each iteration of a Krylov solver. Finally, $x_{I_p}^{(LI)}$ is computed by performing a sparse LU factorization of A_{I_p, I_p} followed by a solve step; both steps are performed using a sparse direct solver. We notice that all the entries of A_{I_p, I_p} are static data allocated on node p because of the block-row mapping.

4.2 Parallel implementation of LSI

The implementation of LSI is slightly more complex because it requires to form $A(:, I_p)$ as well as the computation of the right-hand side of the least squares, which involves a 2-overlap partition. The assembly of $A(:, I_p)$ consists in receiving the off-diagonal block entries of $A(:, I_p)$ that are defined by the edges of $G_p^1 \backslash G_p^0$. This can be implemented without an additional data structure as those required to perform the matrix-vector product. However, the computation of the right-hand side of the least squares involves $G_p^1 \backslash G_p^0$ and $G_p^2 \backslash G_p^1$. In this case, the edges of these subgraphs define the required entries of A while the vertices define the needed entries of x^k to compute $\sum_{q \neq p} A_{:, I_q} x_q^{(k)}$. To summarize, LSI requires two communication steps. The first one to assemble $A(:, I_p)$ and the second one to compute the right-hand side of the least squares problem. Compared to LI, LSI requires a higher volume of communication and possibly, depending on the sparsity pattern of A, a higher number of messages to get the distance two information necessary for the computation of the right-hand side. We notice that this 2-overlap information can be computed at a low extra cost when G^0 and G^1 are built in a pre-processing phase. The final step consists in performing a sparse QR factorization of $A(:, I_p)$ and the associated solve step using the R factor; both steps are performed with a sparse QR solver.

The robustness of LSI comes at the cost of some additional communication with respect to LI. Similarly, some additional storage is necessary to represent $G_p^2 \backslash G_p^1$ for the right-hand side and the matrix that define the local least squares problem. In addition, a sparse QR factorization is significantly more expensive than a sparse LU factorization.

4.3 Parallel assessment

For the numerical experiments reported in this section, we considered the MUMPS package [4] for the solution of the sparse linear systems involved in LI and the qr_mumps package [8] for the least-square solutions in LSI. After a fault, only the replaced node performs the recovery step whose most intensive computation is spent in the sequential sparse factorization. We notice than a more advanced implementation, which is out of the scope of this paper, could involve a few nodes to benefit from the parallel features of the sparse direct solvers.

The main characteristics of the non-symmetric matrices selected for this parallel study are displayed in Table 2. For each of these matrices, we considered parallel experiments varying the

Matrix	Tmt	Audi_unsy	Matrix211	Tdr455k
n	917K	943K	801K	2,738K
nnz	4,5M	77,6M	129,4M	112,7M

Table 2: Description of the matrices considered for the parallel experimentation.

number of computing nodes from 120 to 312. We report in Table 3, the cost associated with the calculation of the recovered iterate after a single node crash. This cost is expressed in terms of GMRES iteration time, which is the elapsed time for the recovery divided by the average time for one parallel GMRES iteration. For those parallel experiments, we consider FGMRES preconditioned with an additive Schwarz scheme [37] with an overlap of one. The local factorizations required for implementing the additive Schwarz preconditioner are performed using the MUMPS sparse direct solver. The recovery time includes the communication time to assemble the local problem (matrix and right-hand side) and the time for its solution using a sparse LU (QR) factorization for LI (LSI, respectively). We do not record the time for recovering the static data, nor the time to allocate a new node after a fault, nor the time to rebuild a coherent MPI communicator. They are identical for LI and LSI. However, they depend on many external components/choices (e.g., storing or recomputing the matrix entries, the future MPI-3 implementation for the fault-tolerant features, the computing platform, ...). This means that the depicted recovery costs are lower bounds of what real recovery would be in practice. In this table, it can be seen that LSI is always substantially more time consuming than LI, the ratio varies from 2.7 to 13.5 in our experiments. We point out that this ratio is an upper bound on what would be observed in practice, if we account for the other recovery costs that we have neglected. This higher cost mainly comes from the fact that a sparse QR factorization is more expensive than a sparse LU factorization even using state-of-the-art solvers [14]. Also, we observed that the time to assemble the rectangular matrix and the right-hand side was actually negligible compared to the factorization time.

Matrix	# nodes	Cost LI	Cost LSI	Ratio
Audi_unsy	120	7	70	10.0
	216	5	58	11.6
	312	4	35	8.8
Matrix211	120	5	40	8.0
	216	4	25	6.3
	312	6	22	3.7
Tdr455k	120	17	229	13.5
	216	6	68	11.3
	312	5	64	12.8
Tmt	120	5	16	3.2
	216	3	8	2.7
	312	3	8	2.7

Table 3: Cost of the recovery expressed in GMRES iteration elapsed time.

The overhead of LSI over LI may be due to multiple factors. One reason if that a sparse QR factorization is significantly more costly than a sparse LU factorization. Another reason is that LSI is performed on a larger system (same number of columns, but more rows) than LI. Noticing that the LI scheme can also be implemented with a sparse QR factorization, we have performed an extra set of experiments consisting in implementing the LI scheme with a sparse QR factorization (using the qr_mumps package again). We have reported the results in Table 4. The third column presents the ratio between the time required to factorize the matrices associated to the LI scheme using either a QR or an LU factorization. The obtained results show that the ratio is significant (between 2.0 and 9.2). On the other hand, the fourth column presents the number of rows ($|J_p|$) and columns ($|I_p|$) of the matrices involved in LSI scheme, the number of columns ($|I_p|$) being

also the order of the (square) linear system associated to the LI scheme. Clearly, the number of columns is not significantly higher than the number of columns. As a conclusion, the extra cost of LSI over LI is mostly due to the method used to solve it (sparse QR factorization) and only slightly increased by the size of the underneath system, which overall explains the ratio between LI and LSI that can be observed in Table 3.

Matrix	# nodes	time(QR)/time(LU)	$(\|J_p\|, \|I_p\|)$
`Audi_unsy`	120	5.7	(9.74e3 , 7.80e3)
	216	6.5	(6.12e3 , 4.33e3)
	312	5.0	(4.67e3 , 2.99e3)
`Matrix211`	120	4.3	(7.87e3 , 6.67e3)
	216	4.6	(4.66e3 , 3.73e3)
	312	2.0	(3.29e3 , 2.54e3)
`Tdr455k`	120	9.2	(2.44e4 , 2.28e4)
	216	8.3	(1.46e4 , 1.26e4)
	312	8.0	(1.11e4 , 8.46e3)
`Tmt`	120	3.2	(7.81e3 , 7.59e3)
	216	2.9	(4.40e3 , 4.22e3)
	312	2.7	(3.07e3 , 2.94e3)

Table 4: Ratio of the QR versus LU factorization time for LI and the dimension of the factorized matrices.

Finally, it can be observed as a general trend that the larger the number of nodes, the lower the recovery cost. This effect is mainly due to the nonlinear cost of the factorization with respect to the size of the local matrices (the larger the number of nodes, the smaller the local problems).

5 Concluding remarks

In this paper, we have investigated some IR techniques to design resilient parallel Krylov subspace methods. The IR techniques are based on simple interpolation ideas that compute meaningful values for the entries of the iterate lost after a node has crashed. Using basic linear algebra arguments, we have shown that for SPD matrices the LI strategy does preserve the A-norm error monotonic decrease of the iterates generated by CG and PCG. We have also shown that the LSI strategy does guarantee the residual norm monotonic decrease generated by GMRES, Flexible GMRES and MINRES as well as for preconditioned GMRES for some classes of preconditioners. For non-SPD matrices, the LI strategy lacks robustness as it might not be defined when the diagonal block involved in its definition is singular.

Because we have considered a restarting procedure after the interpolation phase, we have illustrated the numerical penalty induced by the restarting on short terms recurrence Krylov approaches. For CG, the convergence delay remains acceptable for a moderate number of faults. For GMRES, where a restarting strategy is usually implemented to cope with the computational constraints related to the computation and storage of the orthonormal Krylov basis, the numerical penalty induced by the IR techniques is negligible and can be beneficial in some cases. In addition, the proposed schemes have no overhead when no fault occurs.

We have presented parallel performance of the two approaches to get insight on their respective costs. Mainly due to the higher computational complexity of a sparse QR factorization compared to a Cholesky or LU factorization, the robustness of the LSI strategy comes at a fairly larger cost than LI. Consequently, in practice LI and LSI could be combined based on a decision made at runtime depending on the possible singularity of $A(I_p, I_p)$. Furthermore, to alleviate the computational recovery costs, iterative techniques could also be considered for the solution of the local problem with a stopping criterion threshold related to the accuracy level of the iterate when the fault occurs; such a study will be the focus of a future work.

We assessed the effectiveness of our resilient algorithms by simulating process crashes in a parallel distributed memory environment, considering that the support at system level is out of the scope of this study. One solution could consist in using a system fault tolerant support such as ULFM [38]. Interestingly, these ideas can also be extended to uncorrected bit-flip or more generally to memory corruption when an error detection mechanism is available that provides the location of the corrupted memory. The common assumption is that we have the ability to know what part of data in memory has been lost. We point out that a task-based attempt has been proposed by the BSC group where our IR strategies have been implemented to survive data corruption at the granularity of the memory page [25].

We have experimented a general procedure applicable to any linear solver. It would be worth assessing the proposed interpolation strategies in efficient fixed point iteration schemes such as multigrid, where the penalty associated with the Krylov restarting would vanish [15, 19]. For Krylov solvers, the larger the number of faults, the slower the convergence mainly due to the restart. It will be the focus of future research to tune the IR method for a specific Krylov solver in order to attempt to regenerate more information, in particular on the global Krylov space to alleviate the penalty induced by the restart after each fault. We can mention that such ideas can naturally be applied in the context of some eigensolvers [2] and similar investigations deserve to be undertaken in the context of linear system solutions.

Finally, our numerical resilient strategies can be efficiently combined with existing fault tolerance mechanisms such as checkpoint/restart or ABFT techniques to design low overhead resilient tools for extreme scale calculations.

Acknowledgements

We would like to thank the anonymous referees whose constructive comments enabled us to improve the readability of this paper. Finally, this work was partially supported by the French research agency ANR in the framework of the RESCUE project (ANR-10-BLANC-0301), in particular the PhD thesis of the fifth author was funded by this project; this research also benefited from the G8-ECS project.

References

[1] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. Towards resilient parallel linear Krylov solvers: recover-restart strategies. Research Report RR-8324, INRIA, July 2013.

[2] Emmanuel Agullo, Luc Giraud, Pablo Salas, and Mawussi Zounon. On resiliency in some parallel eigensolvers. Research Report 8625, INRIA, 2015.

[3] Lorenzo Alvisi and Keith Marzullo. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Trans. Softw. Eng.*, 24(2):149–159, February 1998. ISSN 0098-5589. doi: 10.1109/32.666828.

[4] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[5] J. Anfinson and F. T. Luk. A linear algebraic model of algorithm-based fault tolerance. *IEEE Trans. Comput.*, 37:1599–1604, December 1988. ISSN 0018-9340. doi:10.1109/12.9736.

[6] Todd M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 32, pages 196–207. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7695-0437-X.

[7] Anita Borg, Jim Baumbach, and Sam Glazer. A message system supporting fault tolerance. *SIGOPS Oper. Syst. Rev.*, 17(5):90–99, October 1983. ISSN 0163-5980. doi: 10.1145/773379.806617.

[8] Alfredo Buttari. Fine-grained multithreading for the multifrontal QR factorization of sparse matrices. *SIAM Journal on Scientific Computing*, 35(4):C323–C345, 2013.

[9] Franck Cappello, Henri Casanova, and Yves Robert. Preventive migration vs. preventive checkpointing for extreme scale supercomputers. *Parallel Processing Letters*, pages 111–132, 2011.

[10] Zizhong Chen. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *ACM SIGPLAN Notices*, volume 48, pages 167–176. ACM, 2013.

[11] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *j-TOMS*, 38(1):1:1–1:25, November 2011.

[12] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A Survey of Rollback-recovery Protocols in Message-passing Systems. *ACM Comput. Surv.*, 34(3):375–408, September 2002. ISSN 0360-0300. doi:10.1145/568522.568525.

[13] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel. The performance of consistent checkpointing. In *Reliable Distributed Systems, 1992. Proceedings., 11th Symposium on*, pages 39–47. Oct 1992. doi:10.1109/RELDIS.1992.235144.

[14] Alan George and Esmond Ng. On the complexity of sparse qr and lu factorization of finite-element matrices. *SIAM journal on scientific and statistical computing*, 9(5):849–861, 1988.

[15] Dominik Göddeke, Mirco Altenbernd, and Dirk Ribbrock. Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing. *Parallel Computing*, 49:117–135, 2015. doi:10.1016/j.parco.2015.07.003.

[16] John A. Gunnels, Robert A. Van De Geijn, Daniel S. Katz, and Enrique S. Quintana-ortí. Fault-tolerant high-performance matrix multiplication: Theory and practice. In *Dependable Systems and Networks*, pages 47–56. 2001.

[17] M. R. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear System. *J. Res. Nat. Bur. Stds.*, B49:409–436, 1952.

[18] Kuang-Hua Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, 33:518–528, June 1984. ISSN 0018-9340.

[19] Markus Huber, Björn Gmeiner, Ulrich Rüde, and Barbara I. Wohlmuth. Resilience for multigrid software at the extreme scale. *CoRR*, abs/1506.06185, 2015.

[20] R.K. Iyer, N.M. Nakka, Z.T. Kalbarczyk, and S Mitra. Recent advances and new avenues in hardware-level reliability support. *Micro, IEEE*, 25(6):18–29, Nov 2005. ISSN 0272-1732. doi:10.1109/MM.2005.119.

[21] David B Johnson and Willy Zwaenepoel. *Sender-based message logging.* 1987.

[22] Julien Langou, Zizhong Chen, George Bosilca, and Jack Dongarra. Recovery Patterns for Iterative Methods in a Parallel Unstable Environment. *SIAM J. Sci. Comput.*, 30:102–116, November 2007. ISSN 1064-8275. doi:10.1137/040620394.

[23] C.-C.J. Li and W.K. Fuchs. Catch-compiler-assisted techniques for checkpointing. In *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 74–81. June 1990. doi:10.1109/FTCS.1990.89337.

[24] Yudan Liu, R. Nassar, C.B. Leangsuksun, N. Naksinehaboon, M. Paun, and S.L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–9. April 2008. ISSN 1530-2075. doi:10.1109/IPDPS.2008.4536279.

[25] Casas Marc, Bronevetsky Greg, Labarta Jesus, and Valero Mateo. Dealing with faults in HPC systems. In *PMAA-International Workshop on Parallel Matrix Algorithms and Applications*. Lugano, Switzerland, July 2014.

[26] N. Oh, P.P. Shirvani, and E.J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *Reliability, IEEE Transactions on*, 51(1):63–75, Mar 2002. ISSN 0018-9529. doi:10.1109/24.994913.

[27] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numerical Analysis*, 12:617 – 629, 1975.

[28] J. S. Plank, Y. Kim, and J. Dongarra. Fault tolerant matrix operations for networks of Workstations Using Diskless Checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):125–138, June 1997.

[29] James Plank. An overview of checkpointing in uniprocessor and distributed systems, focusing on implementation and performance. Technical report, 1997.

[30] J.S. Plank and K. Li. ICKP: a consistent checkpointer for multicomputers. *Parallel Distributed Technology: Systems Applications, IEEE*, 2(2):62–67, Summer 1994. ISSN 1063-6552. doi: 10.1109/88.311574.

[31] Narasimha Raju, Gottumukkala, Yudan Liu, Chokchai B. Leangsuksun, Raja Nassar, and Stephen Scott2. Reliability Analysis in HPC clusters. *Proceedings of the High Availability and Performance Computing Workshop*, 2006.

[32] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Scientific Computing*, 14:461–469, 1993.

[33] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003. ISBN 0898715342.

[34] Y. Saad and M. H. Schultz. GMRES: A Generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing*, 7:856–869, 1986.

[35] J.C. Sancho, F. Petrini, K. Davis, R. Gioiosa, and S. Jiang. Current practice and a direction forward in checkpoint/restart implementations for fault tolerance. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8 pp.–. April 2005. doi:10.1109/IPDPS.2005.157.

[36] M. Scholzel. Reduced Triple Modular redundancy for built-in self-repair in VLIW-processors. In *Signal Processing Algorithms, Architectures, Arrangements and Applications, 2007*, pages 21–26. Sept 2007. doi:10.1109/SPA.2007.5903294.

[37] B. F. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1st edition, 1996.

[38] Keita Teranishi and Michael A. Heroux. Toward local failure local recovery resilience model using MPI-ULFM. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, pages 51:51–51:56. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2875-3. doi:10.1145/2642769.2642774.

[39] T.N. Vijaykumar, I Pomeranz, and K. Cheng. Transient-fault recovery using simultaneous multithreading. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 87–98. 2002. ISSN 1063-6897. doi:10.1109/ISCA.2002.1003565.

[40] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.

[41] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Hybrid full/incremental checkpoint/restart for MPI jobs in HPC environments. In *Dept. of Computer Science, North Carolina State University*. 2009.

[42] Chris Weaver and Todd M. Austin. A Fault Tolerant Approach to Microprocessor Design. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (Formerly: FTCS)*, DSN '01, pages 411–420. IEEE Computer Society, Washington, DC, USA, 2001. ISBN 0-7695-1101-5.

[43] Wallodi Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18:293–297, 1951.